Upgrading the Data Processing for the Vacuum Furnaces

Jennifer Case
Department of Mechanical Engineering
Northern Illinois University
DeKalb, Illinois, 60115

Supervisor: Mayling Wong
Cavity Processing Department
Technical Division

SIST Program
August 8, 2012

Fermi National Accelerator Laboratory
Batavia, Illinois, 60510

Abstract

Superconducting Radio Frequency (SRF) Cavities go through processing to obtain the high quality factor necessary for use in an accelerator. Hydrogen degassing in a vacuum furnace is one of the many processing steps. However, the data processing at this step was slow due to the manner in which the data was recorded. The data came from two instruments, a Residual Gas Analyser (RGA) and a Programmable Logic Controller (PLC), that worked seperately. The RGA's data was collected via an existing LabVIEW program. This program was modified to collect data from both instruments to help ease the data processing. Updates were made on the program to show a graph of significant real-time data being sent in from both instruments. In upgrading the program a memory leak occured. When the leak was discovered, techniques suggested by National Instruments (NI) were used to fix it [1]. With the data combined into one sheet, Excel macros could be created to format and process the data. This would save time since the processing would no longer have to be done manually. Although, some formatting would still be required after the macros ran. These upgrades to the data processing for the vacuum furnaces efficiently completed the tasks that they were made to do. The data processing for the vacuum furnaces' bakes can now be done much faster than was previously possible; thus, leaving more time for the data to be analysed.

1 Introduction

Before Superconducting Radio Frequency (SRF) cavities can be used in an accelerator, they must go through processing to ensure that they meet the quality expectations. One of the important processing steps these cavities go through is a bake in a vacuum furnace [2]. Data is collected from the furnace for each bake. This data must then be analysed to ensure that nothing went wrong during the bake. If an issue with the cavity can be found during this step in the process, that will save time since the cavity will not be sent into testing with a known flaw; however, the data processing at this stage was slow and needed to be updated for greater efficiency.

1.1 Introduction to the Processing of SRF Cavities

Project X is going to be run by an accelerator with SRF cavities. These cavities can accelerate particles quickly in short distances relative to the RF cavities [3]. The SRF cavities are made from niobium and are manufactured off-site. While these cavities are manufactured within specifications from FermiLab, they still go through processing at the lab. Before the cavities can be used in an accelerator, they must have met or exceeded a specified quality factor Q. The processing that the cavities go through help to bring the quality factor of the cavities up.

The cavities first go to either a centrifugal barrel polishing area or a horizontal electropolishing area. If the cavities go through the centrifugal barrel polishing, they also receives a rinse and some electropolishing. Next, the cavities go through a high pressure water rinse and then into vacuum furnaces for hydrogen degassing. After the furnaces, the cavities get some more electropolishing, another rinse, and then are assembled to be tested [2].

Baking the cavity at temperature up to 750 °C– 800 °C has been shown to increase the quality factor of the cavity. The bake eliminates most of the hydrogen inside the cavity. However, while the cavity cools, it again becomes vulnerable to hydrogen, which would lower the quality factor [4]. Using a vacuum furnace lowers the risk of hydrogen entering the cooling cavity, thus ensuring a high quality factor.

Despite all the processing, flaws may still exist within the cavities. Testing the cavities is done to check if any flaws remain after processing. However, if a flaw can be found during the processing phase rather than in the testing phase of the cavities, time and labor can be saved since the cavities would no longer have to go through assembly and testing to discover the flaw.

During the bake in the vacuum furnace, flaws can be found by observing what gases come off the cavities. A residual gas analyser (RGA) is used to measure the mass of the gases. The RGA measures atomic mass units (AMUs) from one to one hundred. If certain AMUs spike, that can indicate a dirty cavity [5]. If a cavity appears to be too dirty during the bake, the bake could be shut down and the cavity removed to prevent damage to the furnace. The data from all the bakes go through their own processing to see how well the bake went overall. However, this processing could be time consuming.

1.2 Overview of Data Processing Procedure

Two instruments are used to collect the data from the vacuum furnaces. One is a Stanford Research Systems Residual Gas Analyser (SRS RGA), which collects the partial pressures for a given set of AMUs (usually one to one hundred). The other instrument is an Allen Bradley Programmable Logic Controller Small Logic Controller 500 (PLC SLC500), which collects a variety of data including temperatures and pressures inside the furnace chamber.

To do the data processing, information from both instruments is necessary; however, these two devices worked separately. The data from the RGA was being collected into a text file through a standalone application provided by SRS. The data from the PLC was collected into a database file. The instruments collected data on different time schedules, the RGA collecting every minute and the PLC collecting every ten seconds. The data from each device was outputted into two different files with different formats.

	A	AKW	ALC	ALD	ALE	ALF	ALG	ALH	ALI	ALJ
1	SRSRGA Log Fi									Temperature (°C)
2	Time	99.3	99.9	100	Total		Date	Time		Chamber Average
3	5/26/12 7:33	3.68E-10	5.68E-11	7.90E-11	1.05E-06		5/26/2012	00:00:17	5/26/2012 0:00	18.8000011444
4	5/26/12 7:35	1.08E-10	2.03E-10	2.92E-10	5.09E-07		5/26/2012	00:00:23	5/26/2012 0:00	18.8000011444
5	5/26/12 7:36	3.95E-10	4.17E-10	2.44E-10	4.73E-07		5/26/2012	00:00:33	5/26/2012 0:00	18.8000011444
6	5/26/12 7:37	5.62E-10	1.90E-10	1.98E-10	3.29E-07		5/26/2012	00:00:43	5/26/2012 0:00	18.8000011444
7	5/26/12 7:38	3.02E-10	2.65E-10	1.11E-10	3.58E-07		5/26/2012	00:00:53	5/26/2012 0:00	18.8000011444
8	5/26/12 7:39	5.43E-10	2.01E-10	2.08E-10	3.94E-07		5/26/2012	00:01:03	5/26/2012 0:01	18.8000011444
9	5/26/12 7:40	3.00E-10	1.09E-10	1.93E-10	2.91E-07		5/26/2012	00:01:13	5/26/2012 0:01	18.8000011444
10	5/26/12 7:42	3.19E-10	3.30E-10	3.93E-10	3.22E-07		5/26/2012	00:01:23	5/26/2012 0:01	18.8000011444
11	5/26/12 7:43	-8.93E-11	3.08E-10	1.93E-10	3.15E-07		5/26/2012	00:01:33	5/26/2012 0:01	18.8000011444
12	5/26/12 7:44	3.01E-10	5.82E-10	8.12E-10	3.33E-07		5/26/2012	00:01:43	5/26/2012 0:01	18.8000011444
13	5/26/12 7:45	1.70E-10	1.40E-10	2.76E-10	1.61E-06		5/26/2012	00:01:53	5/26/2012 0:01	18.8000011444
14	5/26/12 7:46	4.92E-10	3.91E-11	2.85E-10	8.37E-07		5/26/2012	00:02:03	5/26/2012 0:02	18.8000011444
15	5/26/12 7:48	3.62E-10	3.17E-10	6.05E-11	6.65E-07		5/26/2012	00:02:13	5/26/2012 0:02	18.8000011444
16	5/26/12 7:49	2.25E-10	7.08E-11	1.88E-10	7.41E-07		5/26/2012	00:02:23	5/26/2012 0:02	18.8000011444
17	5/26/12 7:50	2.80E-10	5.17E-10	5.92E-10	6.07E-07		5/26/2012	00:02:33	5/26/2012 0:02	18.8000011444

RGA

PLC

Figure 1: Formatting time issue between RGA and PLC

Processing each bake took approximately an hour. To start, both files have to be combined into one Excel file. Only certain data from the PLC's file needed to be combined with the data from the RGA. Formatting had to be done to make the two data sets compatible, such as making the date and time the same format, as seen in Figure 1, so that both data sets can be graphed together. The pressure reading from the PLC also had to be altered using the following formula:

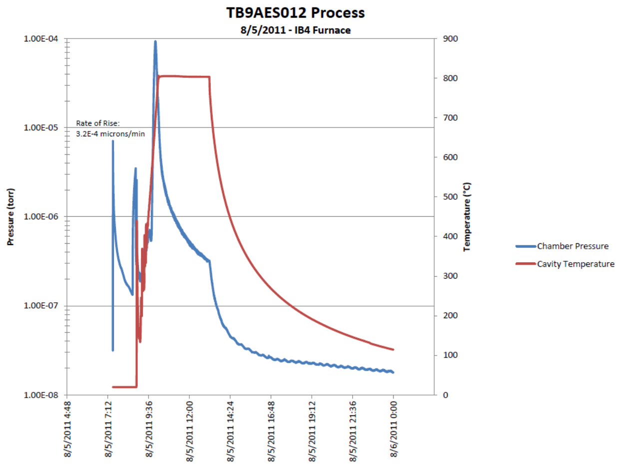
$$\text{Cell} = (\text{POWER}(10,\text{TRUNC}(\text{ALN3})))*(1-(\text{TRUNC}(\text{ALN3})-\text{ALN3}))$$

where ALN3 refers to the cell where the pressure reading from the PLC is located and TRUNC is a function that truncates that number. Figure 2 shows how this formula was used.

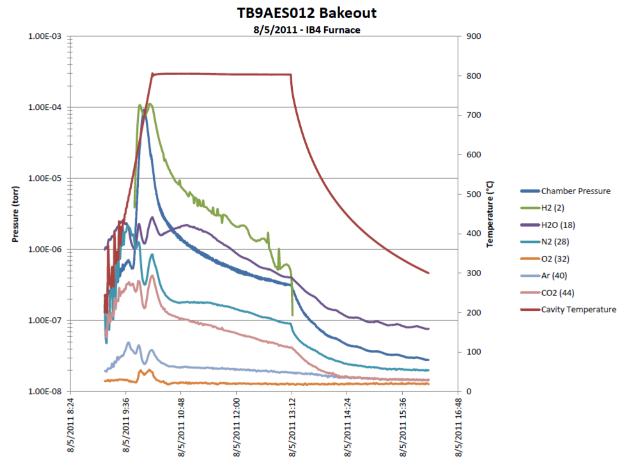
ALO3 f_x =(POWER(10,TRUNC(ALN3)))*(1-(TRUNC(ALN3)-ALN3))

	A	ALE	ALF	ALG	ALH	ALI	ALN	ALO	ALP
1	SRSRGA Log Fi								
2	Time	Total		Date	Time			Pressure (torr)	
3	5/26/12 7:33	1.05E-06		5/26/2012	00:00:17	5/26/2012 0:00	-7.5000004768	5E-08	
4	5/26/12 7:35	5.09E-07		5/26/2012	00:00:23	5/26/2012 0:00	-7.5000004768	5E-08	
5	5/26/12 7:36	4.73E-07		5/26/2012	00:00:33	5/26/2012 0:00	-7.5000004768	5E-08	
6	5/26/12 7:37	3.29E-07		5/26/2012	00:00:43	5/26/2012 0:00	-7.5000004768	5E-08	
7	5/26/12 7:38	3.58E-07		5/26/2012	00:00:53	5/26/2012 0:00	-7.5000004768	5E-08	
8	5/26/12 7:39	3.94E-07		5/26/2012	00:01:03	5/26/2012 0:01	-7.5000004768	5E-08	
9	5/26/12 7:40	2.91E-07		5/26/2012	00:01:13	5/26/2012 0:01	-7.5010004044	4.99E-08	

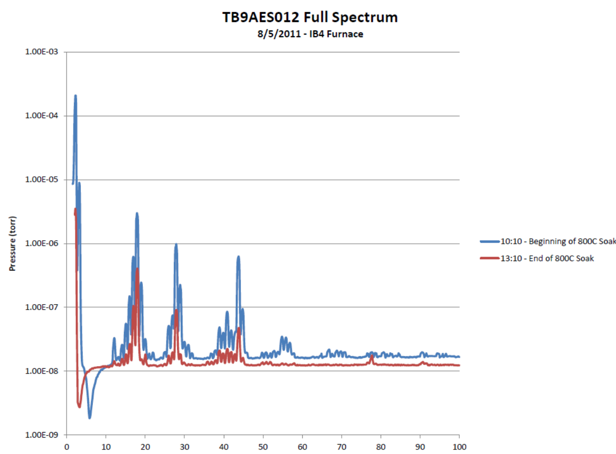
Figure 2: Altering PLC reading for correct pressure



(a) Process Graph



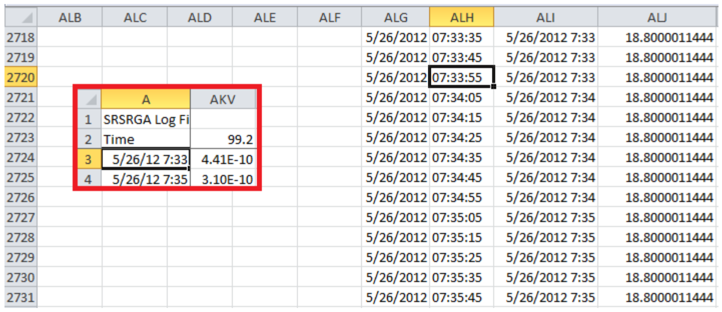
(b) Bake Graph



(c) Spectrum Graph

Figure 3: Graphs from Excel using old method

Once the data was reformatted, three graphs needed to be produced. One graph showed the process of the bake; another showed the bake data of several important AMUs; the last showed the full spectrum of AMUs at both the beginning and end of the soak. Examples of these graphs can be seen in Figure 3. Two out of the three graphs were dependent on data from both the RGA and the PLC in some respects. In order to produce the bake and spectrum graphs, the time-lines of the two data sets needed to be coordinated. As seen in Figure 1, the rows do not line up according to time, so the time must be matched according to the various rows. For example, in Figure 4, 7:33 for the RGA is in row 3, while the same time is in row 2720 for the PLC. For the bake data graph, the times at the start and the end of the graphs must be matched to the given rows for the PLC and the RGA. The formatting and production of these graphs could be sped up if the time-lines matched, which is possible through LabVIEW; the production could be sped up even more using Excel Macros.



	ALB	ALC	ALD	ALE	ALF	ALG	ALH	ALI	ALJ
2718						5/26/2012	07:33:35	5/26/2012 7:33	18.8000011444
2719						5/26/2012	07:33:45	5/26/2012 7:33	18.8000011444
2720						5/26/2012	07:33:55	5/26/2012 7:33	18.8000011444
2721						5/26/2012	07:34:05	5/26/2012 7:34	18.8000011444
2722						5/26/2012	07:34:15	5/26/2012 7:34	18.8000011444
2723						5/26/2012	07:34:25	5/26/2012 7:34	18.8000011444
2724						5/26/2012	07:34:35	5/26/2012 7:34	18.8000011444
2725						5/26/2012	07:34:45	5/26/2012 7:34	18.8000011444
2726						5/26/2012	07:34:55	5/26/2012 7:34	18.8000011444
2727						5/26/2012	07:35:05	5/26/2012 7:35	18.8000011444
2728						5/26/2012	07:35:15	5/26/2012 7:35	18.8000011444
2729						5/26/2012	07:35:25	5/26/2012 7:35	18.8000011444
2730						5/26/2012	07:35:35	5/26/2012 7:35	18.8000011444
2731						5/26/2012	07:35:45	5/26/2012 7:35	18.8000011444

	A	AKV
1	SRSRGA Log Fi	
2	Time	99.2
3	5/26/12 7:33	4.41E-10
4	5/26/12 7:35	3.10E-10

Figure 4: Difference in rows for time matching

1.3 Introduction to LabVIEW

LabVIEW is a graphical programming language that creates Virtual Instruments (VIs) [6]. This means that the code for the program looks similar to flowcharts as seen in Figure 5. The coding for LabVIEW is done on block diagrams, and each block diagram corresponds to a front panel. The front panels can be used to display certain information to the user while the program is running. The user can usually interact with the front panel while the program is running. The information shown in the front panel may be displayed in graphs, charts, arrays, clusters, and more.

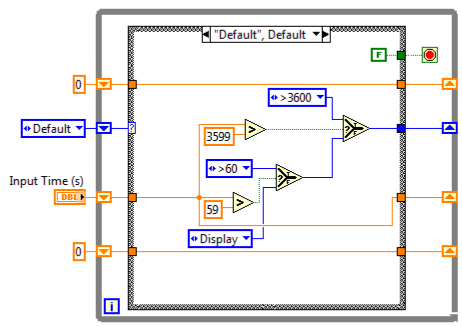


Figure 5: LabVIEW code looks similar to a flowchart

Most complex programs consist of multiple VIs. On these more complicated programs, not all VIs need to be seen by the user. These VIs are called by the main VI of the project, as seen in Figure 6, and are called SubVIs. SubVIs act as functions would in text-based coding. SubVIs can be made to run in the background. The advantage of using SubVIs is the same as using a function in that it cleans up the code.

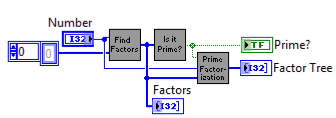


Figure 6: Using SubVIs clean up LabVIEW code

The block diagram has many capabilities common to coding. There are while loops and for loops as well as case structures. While loops are given a condition under which they will stop. For loops have a number wired to them telling them how many times to run; for loops also allow arrays to be sent in and the loops will continue to execute until there are no more elements left in the array, thus eliminating the necessity to wire a number to the for loops. Case structures are often used in the form of state machines. Tutorials on state machines and other useful information can be found on National Instruments (NI) website [7].

Applications can be built in LabVIEW to allow programs to run on computers without LabVIEW.

1.4 Introduction to Excel Macros

If many spreadsheets on Excel go through the same processing, Microsoft helps to ease this processing by allowing the user to produce Macros. Macros are programs that complete certain tasks for the user. There are two ways to make macros: recording macros and using Visual Basic for Applications (VBA), a text-based language, to code them [8].

The easiest way to create a macro is to use the Macro Recorder. To do this, the user starts the recorder, goes through the process that he/she wants done, and then stops the recorder. The user can then run the macro to do the same process over again.

The drawback of using the Macro Recorder is that it is not very intelligent. It will repeat exactly what was done but is unable to figure out what the intentions of the user may have been. This is where the coding with VBA comes in. One advantage of using the Microsoft Visual Basic Editor over the Recorder is in the selection of data. The Editor can be told to select all of the data in a given column, while the Recorder is just told the range to select each time it is run. That can be a problem if data sets of different lengths come in; therefore, it would be more advantageous to use the Editor to intelligently select the data.

The recorder does, however, have its advantages as well. If there is a set of procedures that does not require any intelligent changes, such as the formatting of charts, the recorder presents a simple way to get the code. The code can then be moved to another Macro or the two Macros can be run separately. The most effective way to code Macros is to use both methods.

1.5 Purpose of Project

The more time that is spent processing data from each bake means that less time can be spent on analysing these bakes. A way to simplify and improve the efficiency of this process was necessary if the production speed of the SRF cavities is going to be increased. This could be done with the help of NI's LabVIEW and the Development Kit provided by SRS.

This paper outlines the development of a LabVIEW application and several Excel Macros. The application brings the PLC and the RGA together. The previous timing issue is gone as well as the lengthy reformatting process. Most of the data is written into one text file with the data already processed. The LabVIEW program also displays the bake data graph as it develops over time. As for the production of the graphs, they can now be quickly produced using a set of eight Excel Macros. The Macros are set up to read the data output for the new LabVIEW application.

2 Methods

LabVIEW and Excel Macros were used in the improvement of the data processing for the vacuum furnace.

2.1 LabVIEW

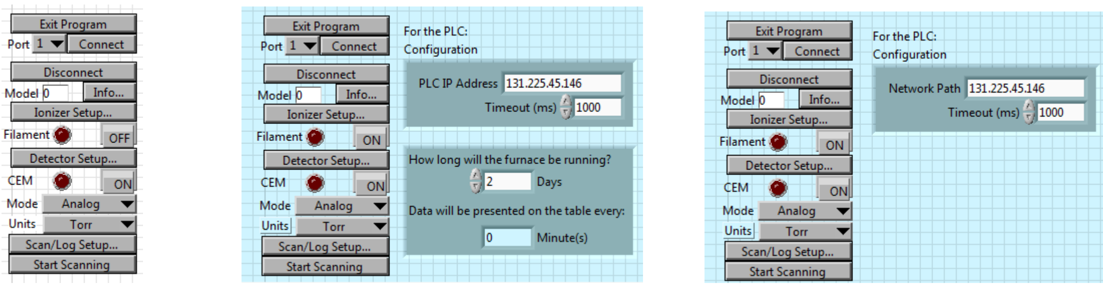
Since SRS provided a Development Kit with their program on it, the main code and structure of the program could be kept while appropriate additions could be made. It was necessary to maintain the Analog graph that was currently being used since it displays real-time data that is used to determine the cleanliness of a cavity [9]; however, it was desired that the LabVIEW program showed the bake chart as time progressed on the bake. This meant that the LabVIEW program would need to take data from both the RGA and the PLC. The PLC would provide the program with the furnace temperatures and pressure.

To ensure that all the data got sent where it was intended and that the log came out properly, there were three VIs that needed to be altered from the original SRS program: *SRSRGAa Main.vi*, *SRSRGAa*

Logging Initialization.vi, and *SRSRGAa Analog.vi*. To communicate to the PLC, *PLC_meas.vi* was added to the program.

Two possible charts were looked at to create the bake chart. One was a XY graph; the other was a waveform chart. The XY graph put the data at accurate times where the waveform chart puts the data at even intervals. However, the XY graph has to be rebuilt every time new data is added while the waveform chart can continue adding data onto the end.

2.1.1 SRSRGAa Main.vi



(a) Original (b) XY (c) Waveform

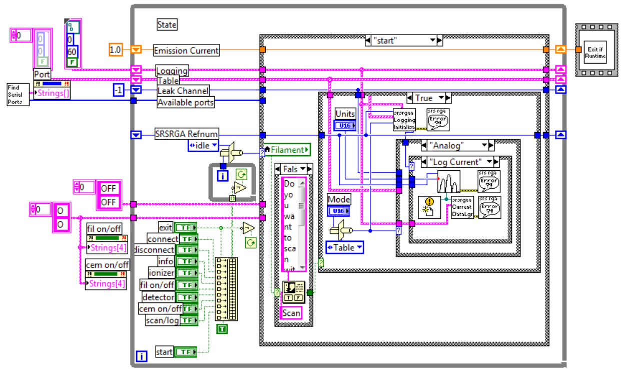
Figure 7: *SRSRGAa Main.vi* Front Panels

SRSRGAa Main.vi is the Startup VI, which means it is the first VI that is seen by the user when the application is run. Figure 7a shows the original *SRSRGAa Main.vi*. From this front panel, the user connects to the RGA by selecting the **Connect** button and disconnect from the RGA by selecting the **Disconnect** button. Other buttons take the user to other VIs where further controls are available. The one that controls the specifications of the scan and the location of the log file is the **Scan/Log Setup...** button. The mode should be set at **Analog** in order for *SRSRGAa Analog.vi* to pop up when **Start Scanning** is selected, which will start the RGA scan.

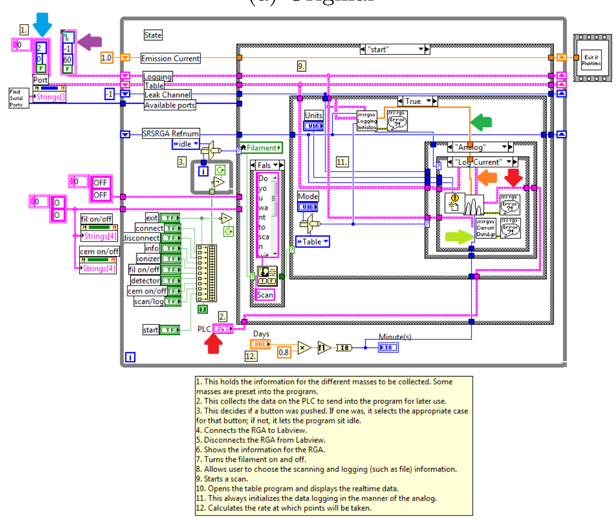
Figure 7b and 7c show the altered front panels. The altered front panels accept information necessary for communication to the PLC, such as the PLC's IP Address. As seen in Figure 7b, the XY application has a third area that asks for the number of days that the furnace will be running. The data that appears on the XY graph will only appear every two or three minutes or so depending on the number of days the program will be running. Only printing every two or three data points helps control the size of the arrays that hold the XY data.

Figure 8 shows the original and altered block diagram codes. Both the waveform and XY code collect data for the PLC, which can be found near the bottom of the while loop indicated by a red arrow. Following the wire from PLC, it can be found to connect up to *SRSRGAa Analog.vi*. This means that the information held within the PLC cluster is sent into the other VI to be used there. There is also more information being fed into *SRSRGAa Analog.vi* than in the original code. The specifications cluster, which holds the masses that need to be plotted on the bake chart, is also sent into the Analog VI, indicated by an orange arrow. The index array from *SRSRGAa Logging Initialization.vi*, which holds the index values of the masses for the bake chart, is also sent in, indicated by a green arrow. A value of minutes for the XY code is also sent into *SRSRGAa Analog.vi*, indicated by a light green arrow.

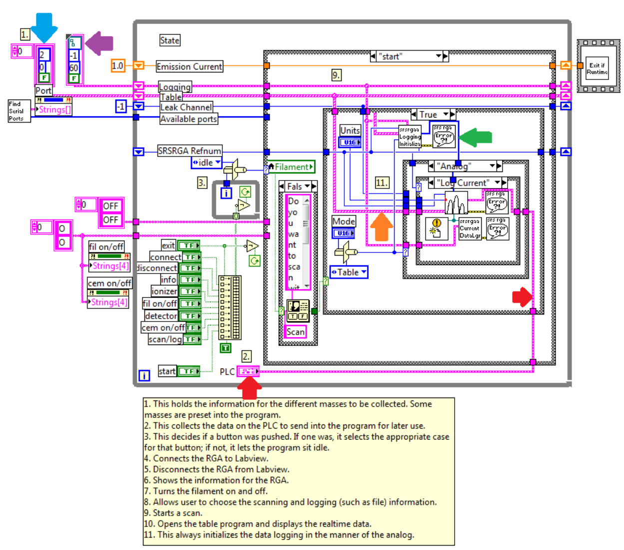
The two pink clusters in the upper right corner of the waveform and XY programs have values initialized in them. The specifications cluster to the left, indicated by a blue arrow, holds the six AMU values that will be plotted; this means that the AMUs will not have to be entered each time the program is run. The logging cluster to the right, indicated by a purple arrow, has the second value as -1, which will automatically set the logging to continue until the program is stopped. This ensures that all the data will be logged.



(a) Original



(b) XY



(c) Waveform

Figure 8: *SRSRGAa Main.vi* Block Diagrams

2.1.2 SRSRGAa Logging Initialization.vi

SRSRGAa Logging Initialization.vi is a subVI that creates the log. This subVI creates the column titles, such as Time, 1, 1.1, 1.2, etc. that is seen at the top of every file created by this program.

Figure 9 shows the original and altered block diagram code. The main difference to the text file is the addition of six columns indicated by a red arrow: Chamber Temperature, Cavity_1, Cavity_2, Cavity_3, Cavity_4, and Chamber Pressure. These columns are and will be filled by information from the PLC.

The new *SRSRGAa Logging Initialization.vi*s also create an index, which is given to *SRSRGAa Analog.vi*. The index is formed using the specifications cluster, which hold the important masses. This is key to pulling the correct AMU readings from the RGA for the bake graph.

In Figure 9b, the coding for the creation of the index lies in a flat sequence structure, indicated by an orange arrow. This means that the first box, indicated by a number 1 in the upper-right corner, in the sequence will always finish before the second box begins. The values array, indicated by green arrows, can be found in both boxes of the sequence, which is why the sequence was used. The values array needs to be filled from the original for loop, indicated by a light green arrow, before it can be properly read. If the flat sequence box was not there, the computer could go through the code in the second box before the values array was filled.

If the Specifications cluster's wire, indicated by blue arrows, is followed, it leads into the second box in the sequence. From there, it goes into the bottom for loop. Specifications is unbundled and the mass is filtered out. The mass is then reformatted as a string and sent out of the for loop where it recollects as an array.

This new array of masses is then sent into another for loop, where it compares with the array of values. The values array is formatted identically to the mass array. Each mass is then searched for among a list of AMUs. When the mass is located in the Values array, the index number for that mass is sent out of the for loop, where it collects into the index array, indicated by a purple arrow. This index array gets sent into *SRSRGAa Analog.vi*.

The code in Figure 9b, however, is not memory friendly and unnecessarily complicated. The values array was stored in two locations since there was a local variable. The index created two copies, one with integers and one with doubles, since the wire did not match the terminal. The exact same thing can be accomplished with the code in Figure 9c. The flat sequence is eliminated since values is directly routed into the for loop. This for loop creates the same index as the other but with greater efficiency. The code in 9c is from the waveform program, but could be moved into the code for the XY program to decrease the memory usage in that program as well.

2.1.3 SRSRGAa Analog.vi

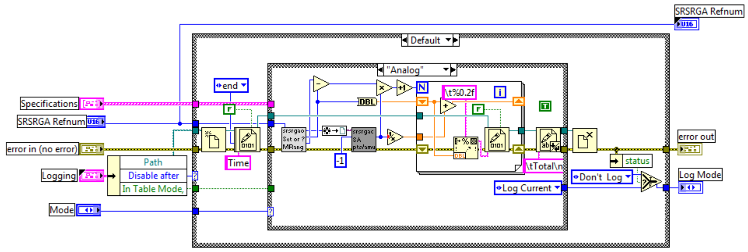
Since the Analog graph existed within *SRSRGAa Analog.vi*, alterations to the VI and the code could be made to display the bake chart as well.

Figure 10 shows the original and altered front panel for the Analog VI. The bake graph was added immediately below the analog graph and programmed to display the proper information.

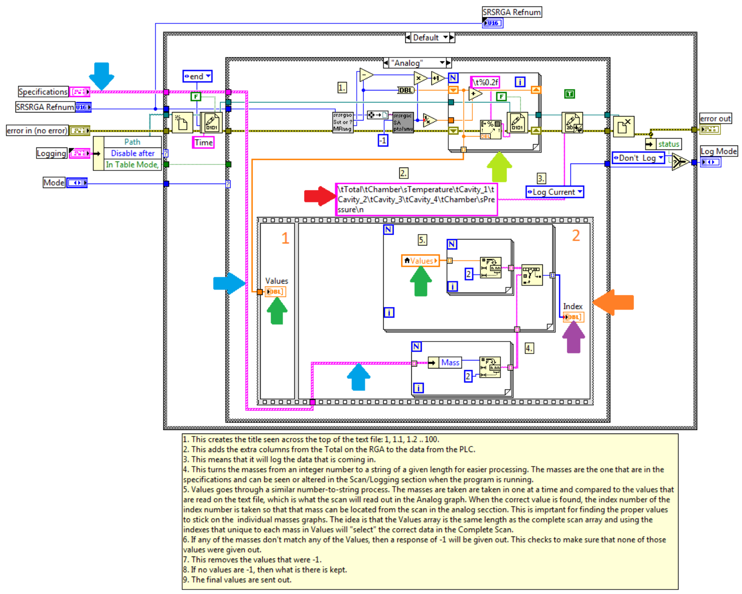
Alterations in the coding within the block diagram of this VI were made to make the bake chart display correctly. Figure 11 shows the original and altered waveform code. The first change in the code can be seen at the top of the altered code, indicated by a red arrow. The first for loop activates fourteen plots. In the next for loop the plots are named according to the masses. Each of these plots are placed on the pressure axis. Subsequent plots are set aside for the five possible temperature readings, which are plotted on the temperature axis. A last plot is activated for the chamber pressure, which is plotted on the pressure axis.

There is also a portion of code at the top, indicated by an orange arrow, that tells the chart to expand according to how much data is on the chart.

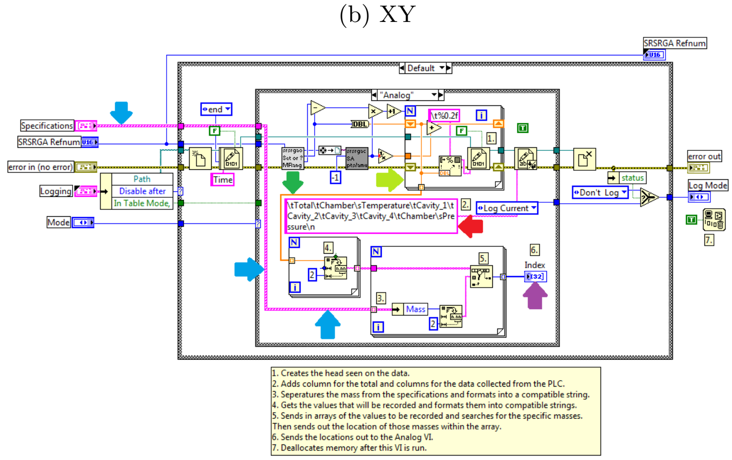
The last change happens within the largest case structure, indicated by a dark green arrow. The case statement is expanded so that every time the analog graph runs through a complete cycle, the bake chart is added to.



(a) Original

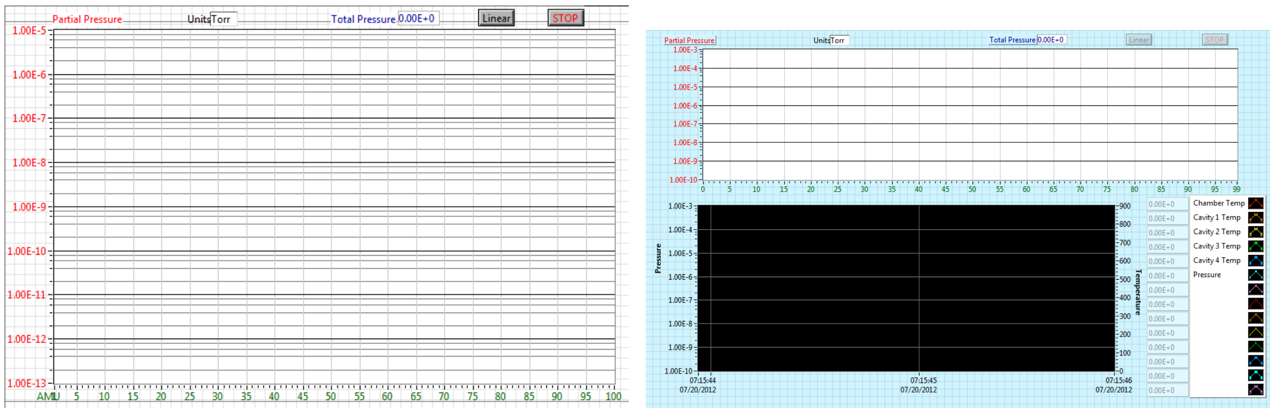


(b) XY



(c) Waveform

Figure 9: *SRSRGAa Logging Initialization.vi* Block Diagrams



(a) Original

(b) XY and Waveform

Figure 10: *SRSRGAa Analog.vi* Front Panels

The Index array, indicated by a light green arrow, is sent into a for loop where the correct value in the Scan array is taken from each index number that is sent in. These values are then compiled into an array with the values from the necessary AMUs.

All the important data from the PLC, such as the temperatures and pressure, are compiled into an array from the *PLC_meas.vi*, indicated by a blue arrow. This array is attached to the array containing the data from the RGA and sent into a notifier to be added into the log file. The data from the PLC is also added to the array of the important AMUs from the RGA. The RGA/PLC array is then taken apart for formatting into the bake chart.

There is a case statement that checks the temperature value inside the furnace, indicated by a purple arrow. It first looks for the temperature to get above 400 °C. Then it looks for the temperature to drop below 180 °C. Once the temperature has dropped below 180 °C, the program stops logging data. However, as long as the program is running, the charts will continue collecting and displaying data.

The waveform chart assumes that the data coming in is collected at even intervals. This, however, is not the case with the RGA data, so there is a bit of inaccuracy in the creation of the waveform chart. The data averages the time difference with how many points there are to best estimate the time in between collections.

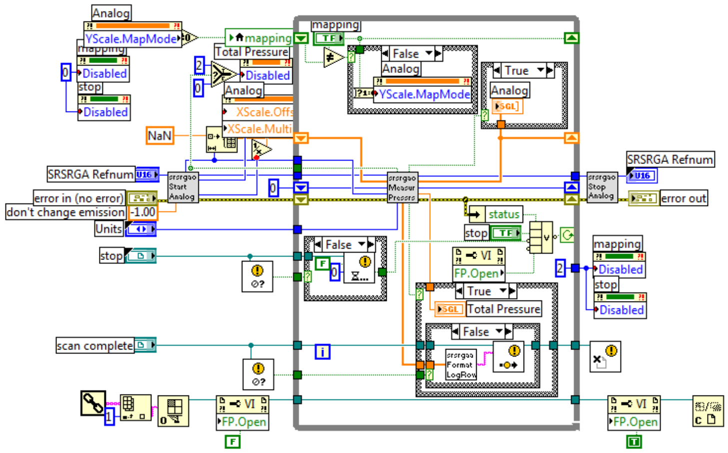
Figure 12 shows the code for the XY program. The XY graph can handle data coming in at uneven intervals, but this plot needs to be recreated every time data is added. The XY graph is the only graph in LabVIEW that allows input on both the X-axis and the Y-axis. To control the X-axis, a timestamp array is used. Shift registers are set up on the while loop to allow data to be added to the array without deleting the information already there. Each of the fourteen potential graphs gets their own numeric array. These arrays are indicated by the blue bracket. Data is added to each from the RGA/PLC array. The X-axis data is bundled with the Y-axis data. All the data from each graph is then combined into one array which gets sent into the bake chart.

The creation of the graph is placed in a case statement, indicated by a burgandy arrow, that only allows data to be added to the plot according to how many data points have passed since the addition of the last data point. This is done to help control the size of the arrays.

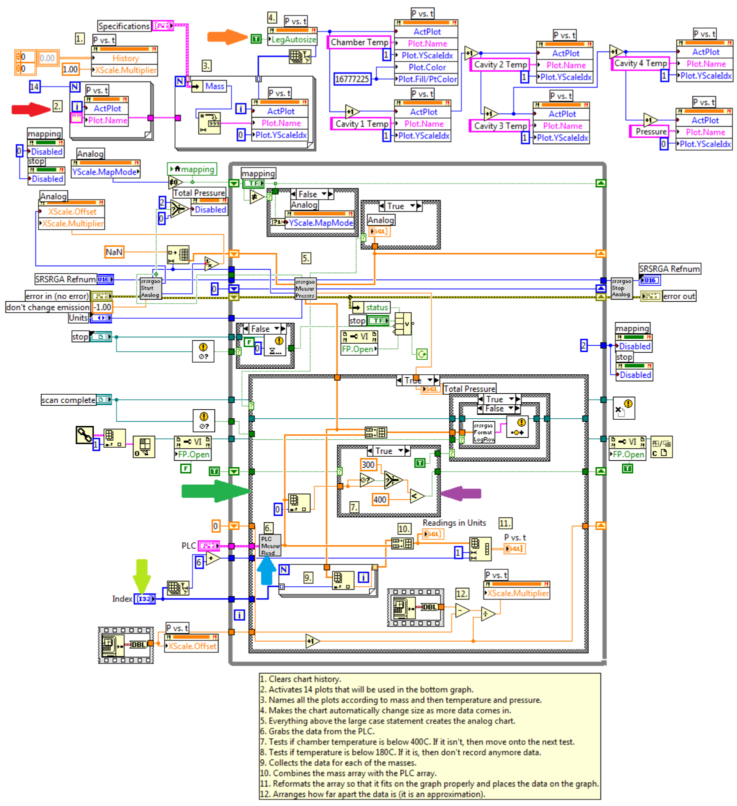
2.1.4 PLC_meas.vi

PLC_meas.vi is a subVI that communicates with the PLC. This subVI pulls off the desired information and then arranges it in an appropriate manner.

Figure 13 shows the code for both the programs; Figure 13a shows the code for the XY program. The PLC cluster is unbundled and sent to three subVIs, indicated by red arrows, that will take the values sent into them and collect the value(s) from the PLC that corresponds with the values sent in. The first two

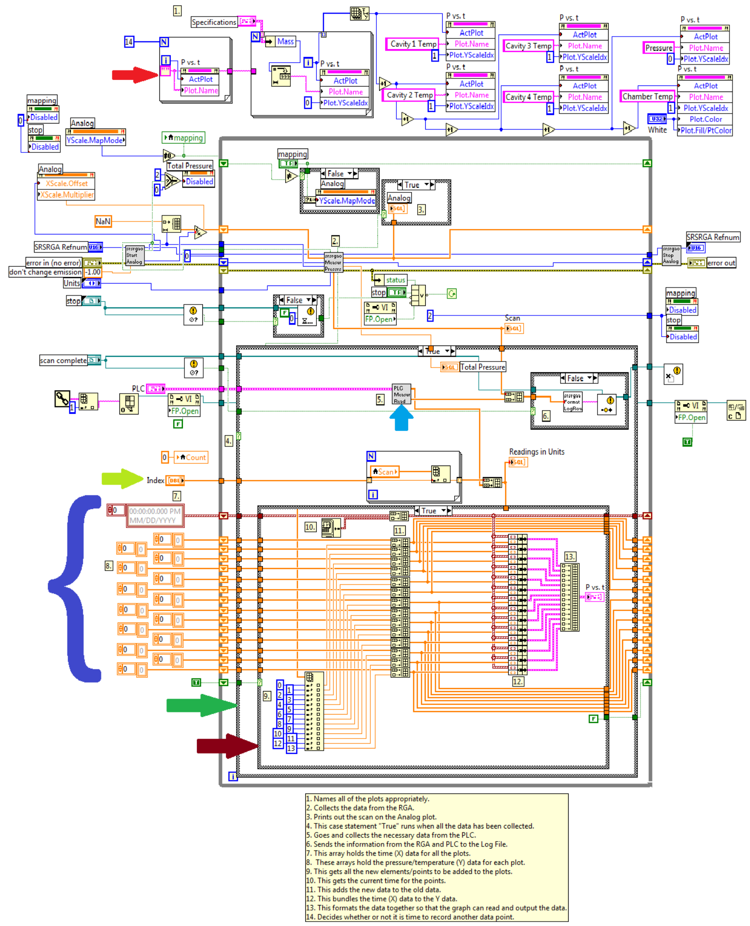


(a) Original

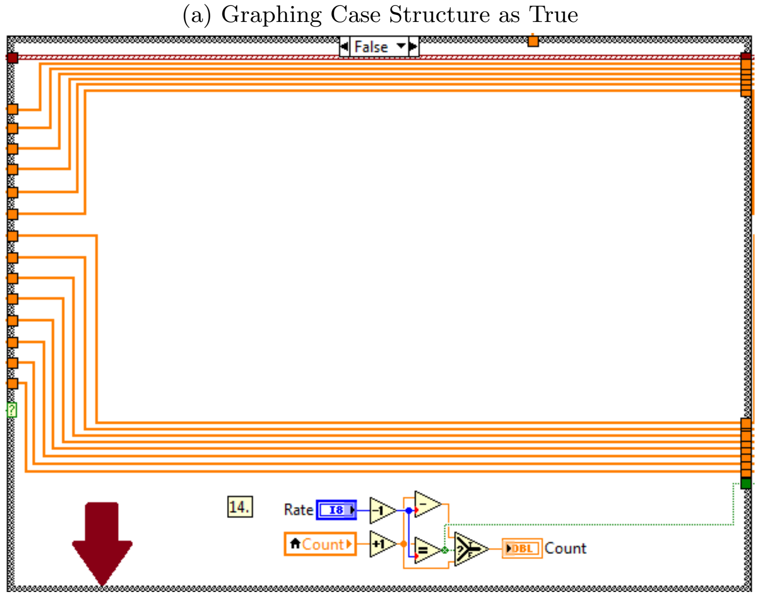


(b) Waveform

Figure 11: *SRSRGAa Analog.vi* Block Diagrams for Original and Waveform Programs

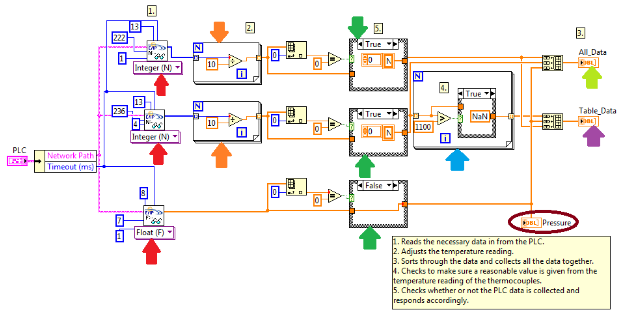


(a) Graphing Case Structure as True

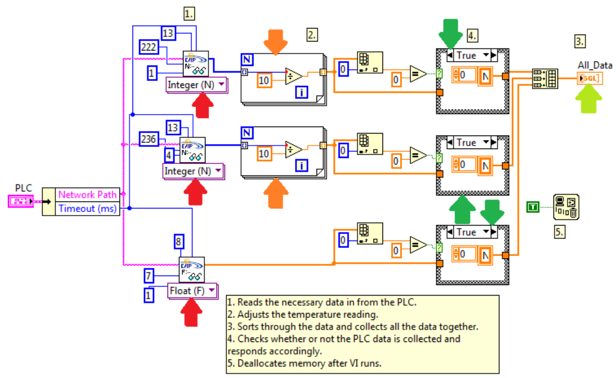


(b) Graphing Case Structure as False

Figure 12: *SRSRGAa Analog.vi* Block Diagrams for the XY Program



(a) XY



(b) Waveform

Figure 13: *PLC_meas.vi* Block Diagrams

values, such as 13 and 222, correspond to an address within the PLC. The last value, such as 1, corresponds to how many consecutive values are to be pulled. The top subVI pulls out the raw data for the chamber temperature. The second subVI pulls out the raw data for the four thermocouple readings. The last subVI pulls out the processed data for the pressure.

Because the temperatures come out in the raw data form, they must be processed before they can be used. The processing for the data is simply to divide it by ten. Each temperature array is sent into a for loop for processing, indicated by orange arrows.

All the data is tested to see if the PLC timed out when collecting the data. This testing occurs in a case statement indicated by green arrows. If the PLC did time out, the filler NaN, which stands for Not-a-Number, is put in place.

All the processed data from each subVI is collected into an All_Data array that will be added to the log file, indicated by a light green arrow.

The thermocouple temperature, however, is only valid if it is less than 1100. If this does not meet that criteria, the filler NaN is used to keep the information off of the bake chart. This occurs inside the for loop indicated by a blue arrow. The data to be added to the bake graph is then sent into the Table_Data array indicated by the purple arrow.

Once again, a waste of memory can be seen in the code in Figure 13a. The code in Figure 13b does essentially the same thing, but saves memory. In Figure 13a, a pressure array, circled in burgandy, is created that holds the pressure data, but is never used. There are also two arrays that hold all the data, therefore creating a redundancy of data. The code in Figure 13b removes that redundancy at the cost of replacing the unnecessary thermocouple readings with NaN. However, those unnecessary readings are recorded at high enough temperatures that they do not appear on the bake graph. The code in 13b is from the waveform program, but could be moved into the code for the XY program to decrease the memory usage in that program as well.

2.1.5 Testing

The testing for these programs was done by running them on the vacuum furnace's computer over an extended period of time, such as four days or so, when time was available. The programs were often run with a bake to see how the program reacted and to produce a common bake chart.

2.2 Excel Macros

The excel macros were made to be used specifically with the data that comes from the new LabVIEW program. When the charts are made from these macros, there is very little formatting that needs to be added. The macros work with the raw data and do all the formatting for the user. Although, the user may have to move the .dbf file from the PLC to a sheet on the file logged from the LabVIEW program.

These macros are user-friendly so that if a user does not know how to use them, the macros will tell what needs to be done.

There are eight macros in total: *A1_CollectInfo_Format*, *A2_chartProcess*, *A3_chartBake*, *A4_chartSpectrum*, *Opt_BakeAdd*, *Opt_SpectrumAdd*, *Opt_RateOfRise*, and *ManualOverride_Information*.

There are also five functions: *GetInfo*, *GetLastrow*, *BakeRange*, *FormatChart*, and *ColToLetter*.

2.2.1 *A1_CollectInfo_Format*

A1_CollectInfo_Format must be run on the sheet containing the RGA data.

This macro will first ask for the name of the cavity and the date that the cavity was baked on. This data is stored publicly since it will go on every chart. The macro then checks to see if there is any PLC data; if there is not, it will create a dummy sheet. The PLC data will then get the date and time formatted to match the RGA data.

In certain bakes, it is necessary to turn the RGA off and pump the pressure up. This causes a loss in the RGA data. The PLC data must then be taken from the original .dbf file and placed correctly into the LabVIEW data. The program has been made to account for this. If there is a gap in the RGA data, it will check and see if there is PLC data available. If there is, the program pulls over the data and formats it accordingly; if there is not, it leaves the line blank to signify the gap in the data.

The macro then averages the temperatures according to what data is available and creates a rate of rise sheet. If there is no PLC information, the rate of rise sheet is not created and the PLC sheet that holds no significant data is deleted.

2.2.2 *A2_chartProcess*

A2_chartProcess creates the Process chart. If there is already a process chart in existence, it will delete it and replace it with a new process chart.

2.2.3 *A3_chartBake*

A3_chartBake creates the Bake chart by asking the user what the range in temperature is for the chart. The only stipulation in this range is that the first value must be larger than the second or else the range plotted will not be the range desired. The bake chart will also replace a pre-existing bake chart.

2.2.4 *A4_chartSpectrum*

A4_chartSpectrum creates the spectrum chart depending on the soak temperature given by the user. The spectrum chart averages ten data points at the start of the soak and ten data points at the end of the soak to be plotted on the chart.

2.2.5 *Opt_BakeAdd*

In the past, it has been necessary to add an additional AMU to the bake chart. *Opt_BakeAdd* was made to allow additions to the bake chart. The user must select the AMU to be added and then run the macro. The temperature range should be recalled and used. This macro may be run multiple times to add any given number of AMUs to the bake chart.

2.2.6 *Opt_SpectrumAdd*

In the past, it has been necessary to add additional temperatures to the spectrum chart. *Opt_SpectrumAdd* was made to allow additions to the spectrum chart. The user must select the temperature to be added and then run the macro. The macro will then ask how many rows before and behind should be added to the average for the temperature. This averaged information will then be added to the spectrum chart. This macro may be run multiple times to add any given temperature to the spectrum chart.

2.2.7 *Opt_RateOfRise*

The rate of rise, which is a measure of how quickly the temperature rises over a given period of time, is usually added to the process chart. *Opt_RateOfRise* has the user select the pressures that are taken into account with the rate of rise before the macro is run. The macro will then highlight the information in a specified color and calculate the rate of rise for the user. A textbox with the information is then added to the process chart.

2.2.8 *ManualOverride_Information*

ManualOverride_Information allows the user to re-enter the information for the charts, such as the name of the cavity and the date of the bake.

2.2.9 *GetInfo*

GetInfo checks to see if there is data stored for the cavity name and date. If there is not data stored, it collects the information from the user.

2.2.10 *GetLastrow*

GetLastrow gets the last row of the data. First, it checks to see if the spectrum macro has been run, which would throw off last row of the data. Then, it collects the last row accordingly.

2.2.11 *BakeRange*

BakeRange checks to see if the bake range has already been given. If it has not, it collects the bake range and finds the necessary locations in the data.

2.2.12 *FormatChart*

FormatChart does basic formatting that all the charts have in common.

2.2.13 *ColToLetter*

ColToLetter turns a number read for a column into the appropriate letter that excel will recognize. This function was found on ozgrid.com and was written by rory [10].

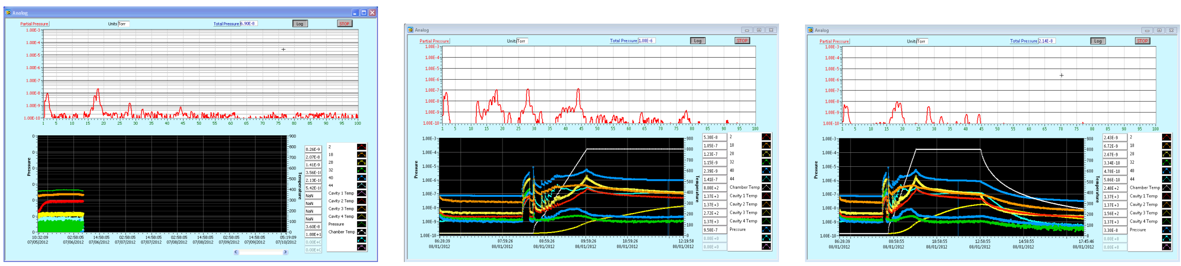
2.2.14 Testing

Once data was collected from the LabVIEW program(s), the macros could be properly tested and debugged on the given excel files.

3 Results

3.1 LabVIEW

The waveform graph originally showed five days on the graph, which was undesirable since the furnace usually runs for about two days, but has been run up to four. To avoid all the data being crammed on one side of the plot area, as seen in Figure 14, an option on the graph allowed it to be expanded over time, similar to the XY plot.



(a) 5 Day Display (b) Scaling Display - Start of Bake (c) Scaling Display - End of Bake

Figure 14: Waveform Chart application

One issue with the waveform plot, as mentioned previously, is that the data is forced to be added on even intervals. This means that the chart is not exactly accurate to the real data, but is more of a good estimate.

The XY graph shows both an accurate depiction of time and always shows the data from beginning to end on either side of the graph. This graph, however, needs to be completely redone every time data is added, which is not the case with the waveform chart. Since there can be thousands of data points in each array, this can slow down the program considerable. On a four to five day trial, the program was slowed by about 30 seconds near the end.

When run, both programs crashed after a given period of time. It was determined that there was most likely a memory leak that was common between both of the programs. The error was likely to be caused by the added code since the original program never had issues with crashing.

Memory from certain VIs, such as *PLC_meas.vi* and *SRSRGAa Logging Initialization.vi*, was deallocated so that it could be used again. All the VIs were cleaned up to avoid unnecessary duplication of data [1]. The new code is now more memory friendly and has yet to crash.

3.2 Excel Macros

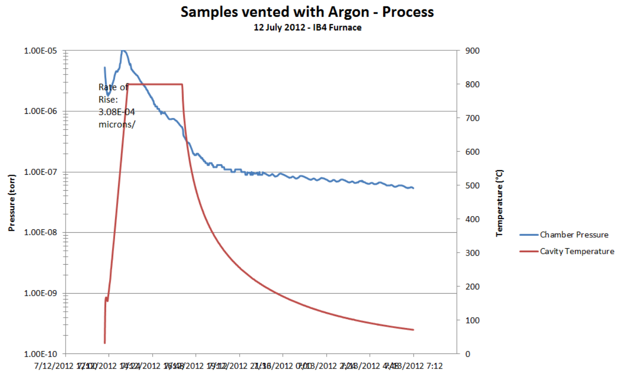
The excel macros cut the data processing time down from around an hour to about five minutes.

Figure 15 shows the graphs produced by the macros initially. Figure 16 shows the graphs produced by the macros after completing the formatting the macros cannot do.

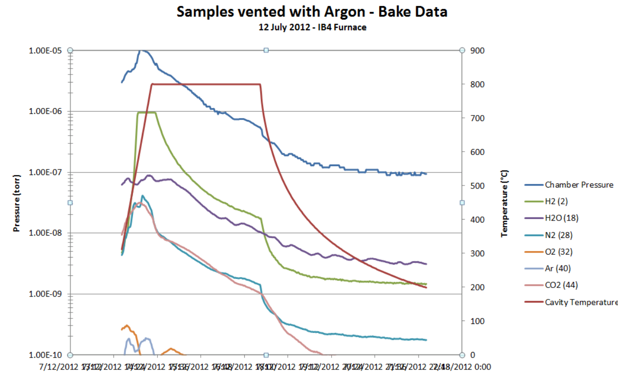
4 Discussion

4.1 LabVIEW

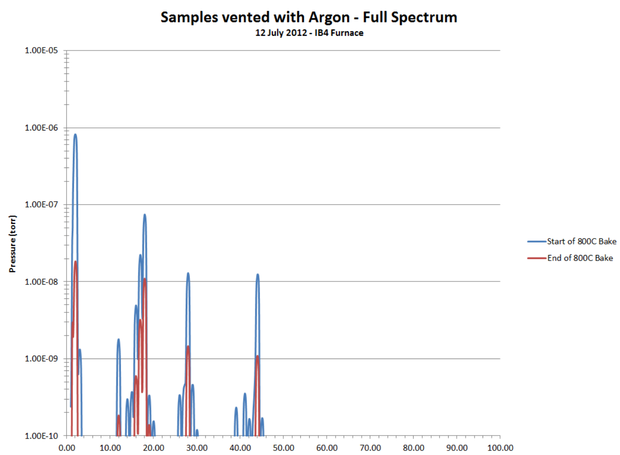
Despite the fact that the memory leak in the programs seems to be patched, the XY program has a chance of crashing regardless.



(a) Process Graph



(b) Bake Graph



(c) Spectrum Graph

Figure 15: Initial graphs created by macros

In order to plot the XY graph, the arrays used to build it continue to expand. There is not unlimited memory to hold the arrays as they continue to grow, so the program could still crash. To help fix this, the program was made to only add data to the graph at given intervals.

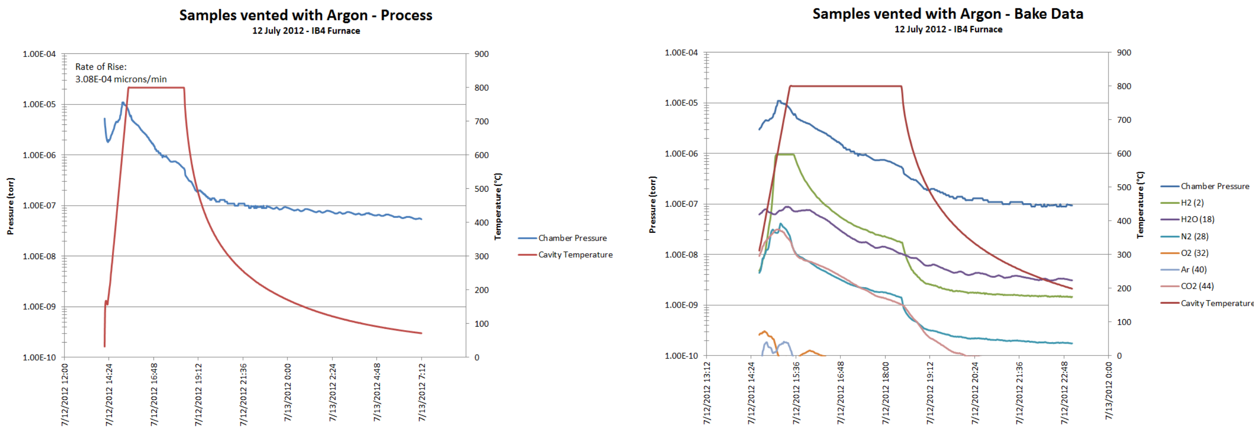
Despite the even spacings of the waveform chart, it is preferable over the XY graph. Both graphs have estimations on them. The waveform has approximate timing; the XY will not have all the data plotted.

The possible crash of the XY graph makes it too unpredictable to be used reliably and building the plot slows down the program, so the waveform graph was chosen instead. The waveform graph shows a good estimation of what is occurring in certain AMUs while the bake is going.

4.2 Excel Macros

The macros save time in the data processing and have become valuable tools.

Both the LabVIEW program and the excel macros offer increased speed and efficiency to the data processing of the vacuum furnace data. These advancements should hopefully help find imperfections in the



(a) Process Graph (b) Bake Graph

Figure 16: Reformatted graphs created by the macros

SRF cavities faster than could be done before.

5 Acknowledgements

I would like to thank Mayling Wong, my supervisor, Nathan Bremer, and Allan Rowe for suggesting improvements for the programs. I would like to thank Diane Engram and all the members of the SIST committee for giving me this wonderful opportunity. I would like to thank Dave Peterson and Luciano Elementi for the help with LabVIEW. I would like to thank my mentors, Dave Peterson and Elmie Peoples for helping out throughout the summer. I would also like to thank James Davenport for giving all the advice about the paper and presentation.

References

[1] Vi memory usage. http://zone.ni.com/reference/en-XX/help/371361H-01/lvconcepts/vi_memory_usage/, June 2011.

[2] C. Cooper, L. Cooley, M. Champion, A. Rowe, O. Pronitchev, V. Poloubotko, and M. Wong. Cavity processing research laboratory at fermilab: Srf cavity processing r&d.

[3] How project x works. http://projectx.fnal.gov/how-project-x-works.shtml, January 2012.

[4] W. Singer. Sc cavities: Material, fabrication and qa. http://www.helmholtz-berlin.de/events/srf2009/programs/tutorials_de.html, September 2009.

[5] Vacuum diagnosis with an rga. http://www.thinksrs.com/downloads/PDFs/ApplicationNotes/Vac_diag_RGA.pdf.

[6] Introduction to labview and computer-based measurements hands-on seminar, January 2010.

[7] Getting started with ni labview student training. http://www.ni.com/white-paper/7466/en.

[8] Microsoft office excel 2007: Vba, 2008.

[9] Nathan Bremer. Conversation, June 2012.

[10] rory. Vba code - return column letter. http://www.ozgrid.com/forum/showthread.php?t=151630, Febuary 2012.